Data pipeline SQL on-premise to Data Lake

Azure Data Factory Pipeline Configuration Details

The file provides a detailed walkthrough of setting up an Azure Data Factory (ADF) pipeline to extract data from an on-premises SQL Server database, transform it, and load it into an Azure Data Lake. Below, I'll summarize the exact configurations, queries, and parameters used in the ADF pipeline for the **source** and **sink** components, as described in the file.

Step1- Setting Up the Azure Environment

1- Create a Resource Group

1-1- Log in to Azure Portal**: Go to portal.azure.com and sign in.**
1-2- Create a Resource Group**:**

- **In the Azure portal, search for** Resource Group **in the search bar.**
- Click **Create.**
- **Fill in the details:**
  - Subscription**: Select your subscription.**
  - Resource Group Name**: xxxx-RG.**
  - Region**: Choose the closest region.**
- **Click Review + Create and then Create.**

2- Create a Storage Account (Azure Data Lake)

2-1- **Search for Storage Account**:

- In the Azure portal, search for **Storage Account**.
- Click **Create**.

Fill in the Details:

- **Subscription**: Select your subscription.
- **Resource Group**: Select the resource group you created (e.g., Xxxx-RG).
- **Storage Account Name**: e.g., Xxxx-SG.
- **Region**: Choose the same region as your resource group.
- **Enable Hierarchical Namespace**: Check this box to enable Data Lake capabilities.
- Click **Review + Create** and then **Create**.

Create Containers:

- Once the storage account is created, go to the storage account.
- Under **Containers**, create the container for raw data.

3- Create a Key Vault

1. **Search for Key Vault**:
    - In the Azure portal, search for **Key Vault**.
    - Click **Create**.
2. **Fill in the Details**:
    - **Subscription**: Select your subscription.
    - **Resource Group**: Select the resource group you created (Xxxx-RG).
    - **Key Vault Name**: Xxxx-KeyVault.
    - **Region**: Choose the same region as your resource group.
    - **Pricing Tier**: Standard.
    - Click **Review + Create** and then **Create**.

4- Create Azure Data Factory

1. **Search for Data Factory**:
    - In the Azure portal, search for **Data Factory**.
    - Click **Create**.
2. **Fill in the Details**:
    - **Subscription**: Select your subscription.
    - **Resource Group**: Select the resource group you created (e.g., Xxxx-RG).
    - **Name**: e.g., Xxxx-DF.
    - **Region**: Choose the same region as your resource group.
    - **Version**: Leave as default.
    - Click **Review + Create** and then **Create**.

Step 2: Setting Up SQL Server and Database

1- Install SQL Server and SSMS

Download SQL Server Express:

- Go to the SQL Server download page and download SQL Server Express.
- Install SQL Server Express on your local machine.

Download SQL Server Management Studio (SSMS):

- Download and install SSMS from the official Microsoft page.

Load the AdventureWorks Database:

- Download the AdventureWorks sample database from the Microsoft website.

2. **Restore the Database**:
    - Open SSMS and connect to your SQL Server instance.
    - Right-click on **Databases** and select **Restore Database**.
    - Choose **Device** and browse to the .bak file you downloaded.
    - Restore the database.

2- Create a SQL User for Data Factory

1. **Create a SQL Login**:
    - Open SSMS and connect to your SQL Server instance.

- Run the following SQL query to create a login and user:

```
CREATE LOGIN username WITH password = '??????';
CREATE USER username FOR LOGIN username;
```

2. **Grant Permissions**:
   - Grant the user access to the AdventureWorks database:

```
USE AdventureWorksDW2019;
GRANT SELECT ON tablename TO username;
```

Step 3: Setting Up Azure Data Factory Pipeline

3.1 Create a Linked Service for SQL Server

1. **Open Data Factory**:
   - Go to the Azure portal and open your Data Factory instance.
   - Click **Author & Monitor** to open the Data Factory UI.
2. **Create a Linked Service**:
   - In the Data Factory UI, go to **Manage** > **Linked Services**.
   - Click **New** and search for **SQL Server**.
   - Fill in the details:
     - **Name**: SQLServer-OnPrem.
     - **Server Name**: Enter your SQL Server name (from SSMS).
     - **Database Name**: AdventureWorksDW2019.
     - **Authentication Type**: SQL Authentication.
     - **Username**: xxxx
     - **Password**: Store the password in Azure Key Vault.
   - Click **Test Connection** and then **Create**.

3.2 Store SQL Credentials in Key Vault

1. **Add Secrets to Key Vault**:
   - Go to your Key Vault in the Azure portal.
   - Under **Secrets**, click **Generate/Import**.
   - Add two secrets:
     - **Username**: xxxx.
     - **Password**: xxxx.
2. **Grant Data Factory Access to Key Vault**:
   - Go to **Access Policies** in Key Vault.
   - Add a new policy:
     - **Principal**: Select your Data Factory.
     - **Secret Permissions**: Grant **Get** and **List** permissions.
   - Save the policy.

3.3 Create a Pipeline to Copy Data

1. **Create a Pipeline**:
   - In the Data Factory UI, go to **Author** > **Pipelines**.
   - Click **New Pipeline** and name it CopyData.
2. **Add a Copy Data Activity**:
   - Drag and drop the **Copy Data** activity into the pipeline.

   - Configure the **Source**:
      - **Dataset**: Create a new dataset for SQL Server.
      - **Table Name**: Select a table (e.g., dbo.DimCustomer).
   - Configure the **Sink**:
      - **Dataset**: Create a new dataset for Azure Data Lake.
      - **File Path**: Set the path to the container.
3. **Debug the Pipeline**:
   - Click **Debug** to test the pipeline.
   - Once successful, click **Publish All** to save the pipeline.

Step 4: Automating the Pipeline

4.1 Create a Trigger

1. **Add a Trigger**:
   - In the Data Factory UI, go to **Author** > **Triggers**.
   - Click **New** and choose **Schedule**.
   - Set the trigger to run daily.
2. **Link the Trigger to the Pipeline**:
   - Select the CopyData pipeline.
   - Save and publish the trigger.

Step 5: Monitoring and Troubleshooting

1. **Monitor the Pipeline**:
   - Go to **Monitor** in the Data Factory UI.
   - Check the status of pipeline runs and troubleshoot any errors.
2. **Check Data in Azure Data Lake**:
   - Go to your storage account and check the container to ensure data is being copied.

---

1. Source Configuration (On-Prem SQL Server)

Linked Service Configuration:

- **Linked Service Name**: SQL Server on-prem
- **Integration Runtime**: Self-hosted integration runtime (since the SQL Server is on-premises).
- **Server Name**: The name of the SQL Server instance (e.g., DESKTOP-XXXXX\SQLEXPRESS).
- **Database Name**: AdventureWorks DW 2019
- **Authentication Type**: SQL Authentication
- **Username**: xxxx (created in SQL Server)
- **Password**: Stored in Azure Key Vault (retrieved dynamically using a linked service to Key Vault).

Source Dataset Configuration:

- **Dataset Name**: SQL Server on-prem
- **Table Name**: Initially, dbo.DimCustomer (later modified to dynamically fetch all tables).
- **Query**:
    - For a single table: SELECT * FROM dbo.DimCustomer
    - For all tables: The lookup activity dynamically generates the query using the schema and table names.

Lookup Activity (to fetch table names):

- **Query**:

  ```
  SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_TYPE = 'BASE TABLE'
  ORDER BY TABLE_SCHEMA, TABLE_NAME;
  ```

  This query retrieves all table names under the dbo schema.
- **Output**: The lookup activity returns a JSON object containing the schema and table names, which is used in the ForEach loop.

2. Sink Configuration (Azure Data Lake Gen 2)

Linked Service Configuration:

- **Linked Service Name**: Azure Data Lake Storage Gen 2
- **Integration Runtime**: Auto-resolve (since the Data Lake is in Azure).
- **Authentication Type**: Default (uses the Azure subscription credentials).
- **Storage Account Name**: The name of the Azure Data Lake Storage account.

Sink Dataset Configuration:

- **Dataset Name**: Parquet sink
- **File Format**: Parquet (column-based format optimized for querying).
- **File Path**:
    - **Container**: name
    - **Directory**: dbo/{table_name}
    - **File Name**: {table_name}.parquet
- **Parameters**:
    - schemaName: Dynamically populated from the ForEach loop (e.g., dbo).
    - tableName: Dynamically populated from the ForEach loop (e.g., Address, Customer, etc.).

Dynamic File Path Configuration:

- The file path and file name are dynamically generated using the following expressions:
    - **Directory Path**:

      ```
      @concat(dataset().schemaName, '/', dataset().tableName)
      ```
    - **File Name**:

      ```
      @concat(dataset().tableName, '.parquet')
      ```

3. Pipeline Configuration

Pipeline Name: Copy All Tables
Activities:

1. **Lookup Activity**:
    - Fetches the list of tables from the SQL Server database.
    - Outputs a JSON object with schema and table names.
2. **ForEach Activity**:
    - Iterates over the list of tables returned by the Lookup activity.
    - **Items**: @activity('Lookup1').output.value
    - **Activities Inside ForEach**:
        - **Copy Data Activity**:
            - **Source**:
                - **Dataset**: SQL Server on-prem
                - **Query**:

                  SELECT * FROM @{item().schema_name}.@{item().table_name}
                    - This query dynamically selects data from each table in the dbo schema.
            - **Sink**:
                - **Dataset**: Parquet sink
                - **Parameters**:
                    - schemaName: @item().schema_name
                    - tableName: @item().table_name
                - **File Path**: Dynamically generated as described above.

4. Key Parameters and Dynamic Content

Dynamic Content in Source Query:

- The source query dynamically selects data from each table using the schema and table names from the ForEach loop:

  SELECT * FROM @{item().schema_name}.@{item().table_name}

Dynamic Content in Sink File Path:

- The file path and file name in the sink are dynamically generated using the following expressions:
    - **Directory Path**:

      @concat(dataset().schemaName, '/', dataset().tableName)
    - **File Name**:

      @concat(dataset().tableName, '.parquet')

5. Key Points to Note

- **Lookup Activity**: Used to fetch the list of tables from the SQL Server database.
- **ForEach Activity**: Iterates over the list of tables and performs the copy operation for each table.
- **Dynamic Content**: Used extensively to dynamically generate queries, file paths, and file names based on the schema and table names.

- **Parameters**: schemaName and tableName are used to dynamically configure the sink dataset.

6. Example of Full Pipeline Flow

1. **Lookup Activity**: Fetches all table names under the dbo schema.
2. **ForEach Activity**: Iterates over the list of tables.
   - For each table, the **Copy Data Activity**:
     - **Source**: Dynamically generates a SELECT * FROM {schema}.{table} query.
     - **Sink**: Dynamically generates the file path and name in the Data Lake (e.g., container/dbo/Customer/Customer.parquet).
3. **Output**: All tables are copied from the SQL Server database to the Azure Data Lake in Parquet format.

This configuration ensures that the pipeline is fully automated and can handle any number of tables in the dbo schema. The use of dynamic content and parameters makes the pipeline flexible and scalable.

Conclusion

This tutorial walks you through setting up a data pipeline using SQL Server, Azure Data Factory, and other Azure services. You've learned how to:

- Set up a Resource Group, Storage Account, and Key Vault in Azure.
- Create and configure a Data Factory pipeline to copy data from an on-prem SQL Server to Azure Data Lake.
- Automate the pipeline using triggers.